بسمه تعالی

تمرین سری دوم درس یادگیری عمیق

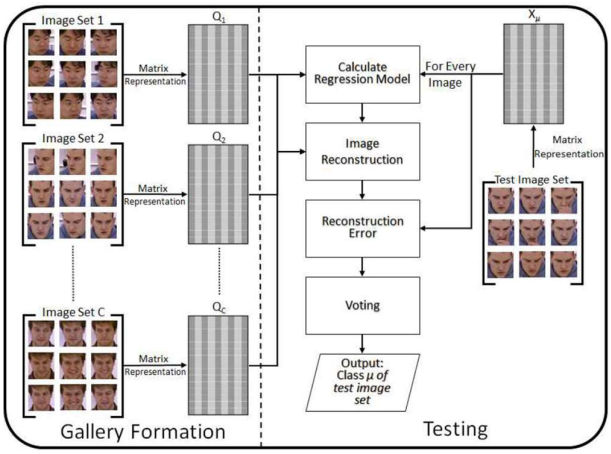
پوریا محمدی نسب

۴۰۰۷۲۲۱۳۸

سوال ۱.

۱- الف) بخش روش پیشنهادی مقاله ۱ را مطالعه نموده و خلاصه ای از آ نها را در گزارش خود بنویسید. (۲۰ نمره)

در این مقاله یک الگوریتم غیرپارامتری(non-parametric) برای کلاسبندی تصاویر معرفی شده است. این تکنیک بر پایه ی بازسازی تصاویر با استفاده از کلاسبندی رگرسیون خطی(linear regression classification) است که به اختصار LRC گفته میشود. ایده ی این الگوریتم این است که تصاویر مربوط به یک دسته(category) با هم یک زیرفضا در یک فضای با ابعاد بزرگ ایجاد میکنند. در زمان آزمون هر تصویر عضو مجموعه تصاویر آزمون(test image set) به صورت ترکیبی خطی از هر مجموعه تصویر گالری ارائه میشود. یک راه حل بر پایه ی حداقل مربعات(least square) برای تخمین پارامترهای مدل رگرسیون برای هر تصویر عضو مجموعه تصاویر آزمون استفاده میشود. در واقع این مدل رگرسیون استفاده میشود تا تصاویر آزمون در گالری تصاویر را بازسازی کند. شاخص و معیار فاصله در این الگوریتم، فاصله اقلیدسی بین تصویر اصلی و تصویر بازسازی شده است. تصویر زیر مراحل گفته شده را به شکلی مصور بیان میکند.



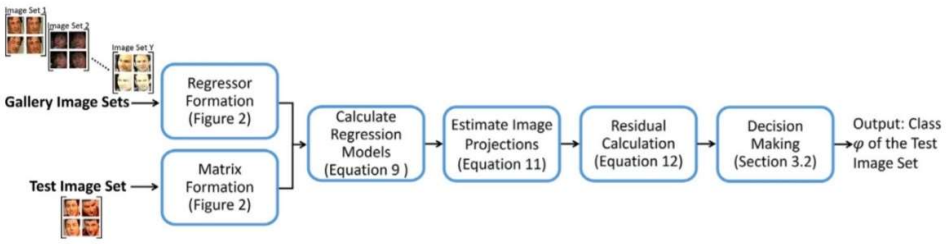
اهداف و دستاورد های این الگوریتم پیشنهادی به صورت خلاصه بیان شده است:

۱) کلاسبند رگرسیون خطی (LRC) جدیدی که پیشنهاد شده است میتواند عملکرد خوبی در شرایط رزولوشن کم تصاویر و همچنین حجم دیتاست کم داشته باشد.

۲) در این نوع روش ها که روی هر تصویر آزمون تعداد زیادی عملگرد اعمال میشود که باعث کاهش سرعت میشود و استفاده از این الگوریتم را برای کاربرد بلادرنگ دچار مشکل میکند. اما این الگوریتم با پیاده سازی موثر ماتریس LRC بهترین سرعت را در بین سایر روش ها دارد.

۱- ب)(امتیازی) برای بهبود عملکرد رو ش پیشنهادی در مقاله بخش الف چه راهکارهای قابل انجامی وجود دارد؟ چرا؟ (در صورت استفاده از منبع حتما ذکر شود)(۱۰ نمره)

در مقاله ی [1] روشی پیشنهاد شده است که شباهت بسیار زیادی به مقاله اصلی سوال دارد و حتی میتوان اظهار کرد که نسخه ی بهبود یافته الگوریتم ذکر شده است. در این مقاله نیز بر اساس یک روش مبتنی بر رگرسیون خطی، تصاویر کلاسبندی میشوند اما بر خلاف مقاله اصلی، اصلا به فاز training نیازی ندارد. نتایج مقاله بهبود یافته روی تصاویر با رزولوشن کم و تصاویر نویزی نشان میدهد الگوریتم بهبود یافته به دیتای بسیار کمتری نیاز دارد و همچنین زمان اجرای کمتری دارد و به خصوص تفاوت این الگوریتم بهبود یافته با الگوریتم اصلی در شرایط چالشی (دیتای کمتر، رزولوشن کمتر) بیشتر مشهود خواهد بود. شکل زیر الگوریتم معرفی شده را نمایش میدهد.



سوال ۲.

در این سوال شبک های را بررسی میکنیم که در لایه آخر از Softmax و تابع ضرر Cross entropy استفاده کرده است. خروجی این شبکه $\hat{y}$ و خروجی مورد انتظار شبکه که به صورت one-hot کد شده است نیز y نام دارد. تابع ضرر Cross entropy و Softmax به صورت زیر تعریف میشود.

$$E(\hat{y}) = CE(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

$$\hat{y} = softmax(o), \hat{y}_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$$

۲- الف) ثابت کنید که اگر به تمام ورودی های Softmax مقدار ثابتی اضافه کنیم، مقادیر خروجی آن تغییری نمیکند. (۱۰ نمره)

فرض کنید ورودی تابع softmax یک وکتور n تایی از o_1 تا o_n باشد.

$$O = \{o_1, o_2, \ldots, o_n\}$$

که در این صورت خروجی تابع softmax با رابطه ی زیر بدست میاید:

$$\hat{y} = Softmax(o)_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$$

حال اگر یک مقدار ثابت k به تمام اعضای O اضافه کنیم داریم:

$$O' = \{o_1 + k, o_2 + k, \ldots, o_n + k\}$$

به صورت مشابه حالت قبل، این ورودی جدید را در فرمول محاسبه softmax قرار میدهیم.

$$\hat{y} = Softmax(o')_i = \frac{e^{o_i+k}}{\sum_j e^{o_j+k}} = \frac{e^{o_i}e^{k}}{\sum_j e^{o_j}e^{k}} = \frac{e^{o_i}e^{k}}{e^{k}\sum_j e^{o_j}} = \frac{e^{o_i}}{\sum_j e^{o_j}}$$

ملاحظه میشود با کمی ساده سازی و خارج کردن e^k از داخل سیگما میتوان این عبارت را از صورت و مخرج خط زد و به همان خروجی حالت قبل رسید.

۲ – ب) یکی از توابع ضرر معروف Hinge نام دارد که رابطه آن در زیر آورده شده است و به صورت مستقیم بر روی خروجی لایه خطی قابل اعمال است.

$$L = \sum_{j \neq i} [\max(0, o_j - o_i + 1)]$$

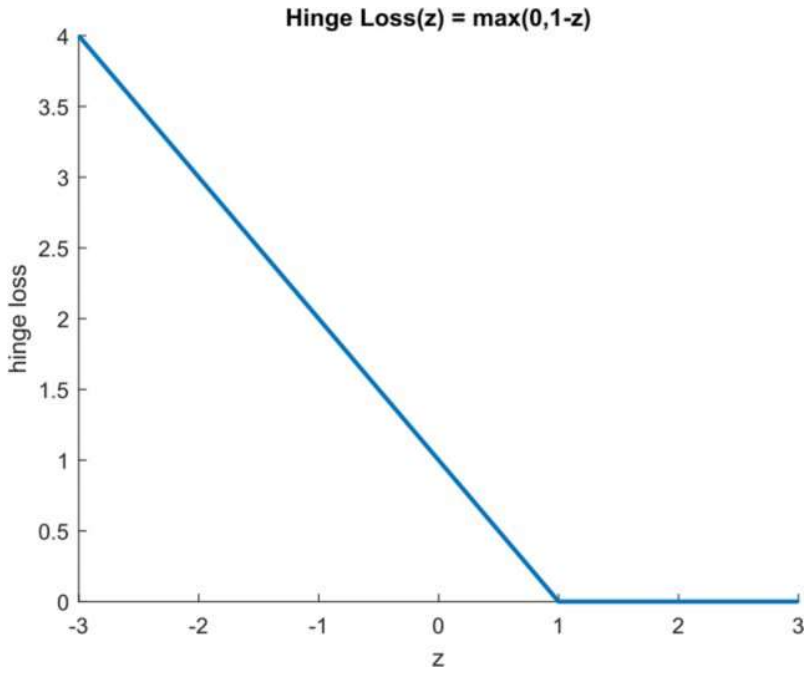
در صورت استفاده از این تابع، بردار گرادیان خطا نسبت به بردار o را محاسبه کنید و با بردار گرادیان حاصل از تابع فعالسازی Softmax و تابع ضرر CE مقایسه کنید.(۲۰ نمره)

$$l(y, \hat{y}) = \sum_{i \neq j} max(0, o_j - o_i + 1)$$

$$l(y, \hat{y}) = \begin{cases} 0 & ; \ True\ predicion \\ o_j - o_i + 1 & ; \ O.w \end{cases}$$

$$\partial_{o_i} l = \begin{cases} 0 & ; \ True\ predicion \\ -1 & ; \ O.w \end{cases}$$

رابطه بدست آمده را میتوان به راحتی با شکل تابع ضرر hinge نیز نشان داد. در حالتی که حدآستانه این تابع(ρ) را برابر یک قرار دهیم اگر پیشبینی کلاسبند درست باشد (بعد از مقدار ρ=1) شیب تابع ضرر صفر است و در غیر این صورت تابع ضرر hinge دارای یک شیب 1- است. منفی بودن این عدد به این معناست که هر چه به ρ=1 نزدیک شویم loss کمتری داریم.



سوال ۳.

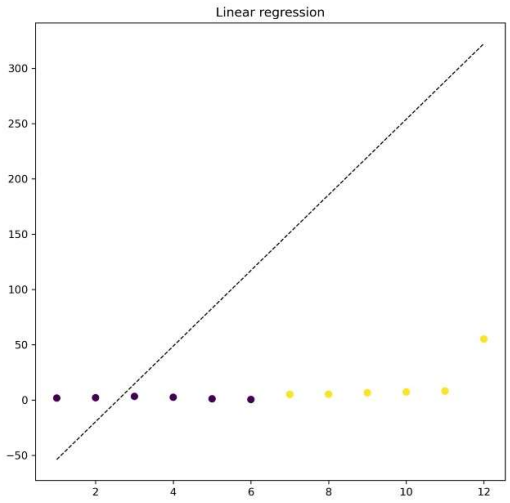
مجموعه داده زیر را در نظر بگیرید که مربوط به یک مسئله باینری است و ورودی آن یک بعدی است. با کمک شبیه سازی و استفاده از دو روش Linear regression و Logistic regression مرز تصمیم میان دو کلاس را یک بار محاسبه کنید. (در این مثال مرز تصمیم x ای است که خروجی مدل برابر با 0.5 باشد). نتیجه بدست آمده را به طور دقیق تحلیل کنید و دو روش را مقایسه کنید. (۲۰ نمره)

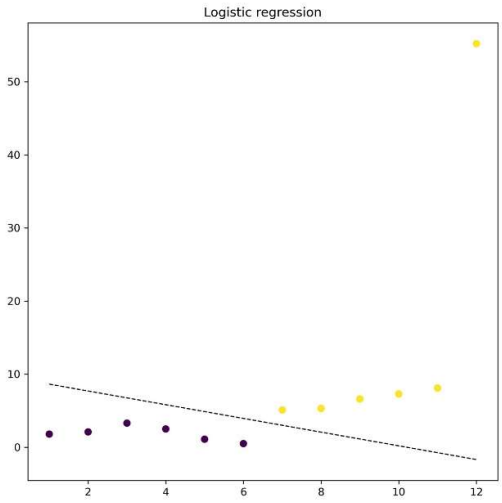
برای حل این سوال در ابتدا کتابخانه های مورد نیاز را فراخوانی میکنیم. در قدم بعدی باید داده های داخل سوال را به شکل آرایه هایی از جنس np.ndarray ذخیره کنیم تا مدل ها بتوانند به راحتی روی آنها کار کنند. نکته ای که وجود دارد این است که هر دو مدل ذکر شده در سوال (رگرسیون خطی و لاجستیک) برای داده های یک بعدی ارور میدهد پس نیاز بود تا index اعداد را نیز به آنها اضافه کنیم تا از وجود خطا جلوگیری کنیم. همچنین اضافه کردن index به دیتا کمک زیادی به مصور سازی داده ها میکند.

مرحله بعدی پس از اماده کردن دیتا، ساخت مدل یادگیری و فیت کردن دیتا درون آن است. به دلیل حجم بسیار کم دیتا عمل train دیتا که با استفاده از تابع fit(x,y) انجام میشود بسیار سریع انجام میشود.(فرایند train کردن برای دیتاست های بسیار بزرگ میتواند مدت بسیار زیادی به طول بیانجامد.).

پس از train شدن مدل، با استفاده از تابع score(x,y) دقت مدل را روی داده های آموزشی اندازه میگیریم. برای مدل رگرسیون خطی این دقت به 0.7636 و برای مدل رگرسیون لاجستیک 1.0000 است.

در ادامه با استفاده از پارامترهای مدل train شده، پارامتر هایی خط تصمیم را محاسبه میکنیم و با استفاده از رابطه خط y = mx + c میتوان خط تصمیم را برای هر مدل روی دیتا رسم کرد. شکل زیر خروجی مصور هر دو مدل را نمایش میدهد. (سورس کد مربوط به این تمرین با نام Deep_learning_HW2_Q3 ضمیمه شده است.)





سوال ۴.

به نوت بوک داده شده مراجعه کرده و بخشهای خواسته شده را تکمیل نمایید. سپس در گزارش خود به تفسیر آن بخشهای بپردازید.(۴۰ نمره)

هدف از این انجام این سوال، طراحی و پیاده سازی یک شبه عصبی با استفاده از Pytorch برای حل مسئله کلاسبندی معروف MNIST است. در ابتدا کتابخانه ها فراخوانی و دیتاست بارگزاری میشود.

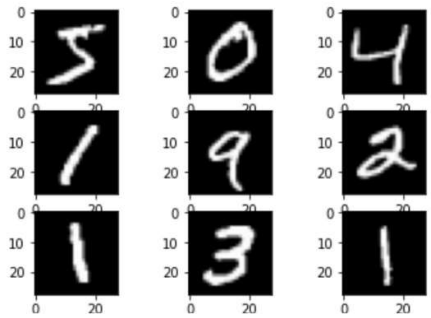
```
[1]: from torchvision import datasets as dt
     import matplotlib.pyplot as plt
     from torchvision import transforms
     import torch
     import torch.nn as nn
     import torch.optim as optim
```

```
[2]: # Load the training data
     mnist = dt.MNIST('data', train=True, download=True)
     mnist = list(mnist)[:2500]
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to data/MNIST/raw/train-images-idx3-ubyte.g
z
  0%|          | 0/9912422 [00:00<?, ?it/s]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to data/MNIST/raw/train-labels-idx1-ubyte.g
z
  0%|          | 0/28881 [00:00<?, ?it/s]
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to data/MNIST/raw/t10k-images-idx3-ubyte.gz
  0%|          | 0/1648877 [00:00<?, ?it/s]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw
```

پس از بارگزاری دیتاست میتوان آن را نمایش داد تا دید بهتری از مسئله پیدا کرد.

```
[3]: # plot the first 9 images in the training data
     import matplotlib.pyplot as plt
     for k, (image, label) in enumerate(mnist[:9]):
       plt.subplot(3, 3, k+1)
       plt.imshow(image, cmap='gray')
```



در ادامه با انجام یک سری از پردازش ها، دیتا را برای آموزش دادن مدل آماده میکنیم. این پردازش ها شامل تبدیل نوع دیتا به tensor ، تبدیل عکس ها از حالت ماتریس ۲۸*۲۸ به یک ماتریس خطی و در نهایت تقسیم دیتاست به دو قسمت train و validation.

[4]:
```
# transform the image data type to tensor
img_to_tensor = transforms.ToTensor()
```

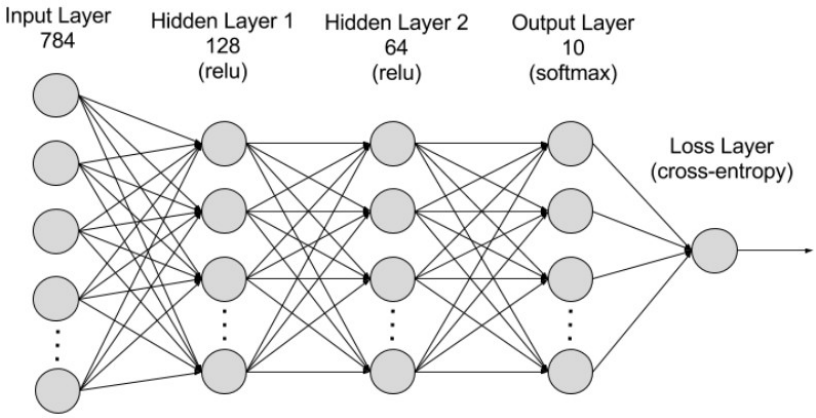
[5]:
```
# convert the last image we saw into a tensor
img_tensor = img_to_tensor(image)
img_tensor.shape
```

[5]:
```
torch.Size([1, 28, 28])
```

[6]:
```
# load the training data as tensor
mnist_train = dt.MNIST('data', train=True, transform=img_to_tensor)
mnist_train = list(mnist_train)[:2500]
```

[7]:
```
#split data into training and validation
mnist_train, mnist_val = mnist_train[:2000], mnist_train[2000:]
```

پس از آماده سازی کامل دیتا، مدل شبکه عصبی مورد نظر را پیاده سازی میکنیم. شکل زیر معماری شبکه را نمایش میدهد.



[11]:
```
############
# Your code
# Start
input_size = 784
hidden_sizes = [128, 64]
output_size = 10

model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0], hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size),
                      nn.LogSoftmax(dim=1))

print(model)
# End
```

پس از طراحی مدل به سراغ تابع اصلی آموزش دهنده میرویم که در سورس کد با نام run_gradient_descent تعریف شده است.

[12]:
```
def run_gradient_descent(model,batch_size=64,learning_rate=0.01,weight_decay=0,num_epochs=10):
  #define the best optimizer and loss function
  ############
  # Your code
  # Start
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.SGD(model.parameters(), lr=0.003, momentum=0.9)
  # End
```

در قطعه کد بالا تعریف تابع و آرگومان های آن را مشاهده میکنیم. بعلاوه دو مورد مهم در این قسمت از کد تعریف شده اند. یکی نوع loss function و دیگری نوع بهینه ساز مسئله است. به دلیل نوع مسئله که یک مسئله کلاسبندی است از تابع ضرر

CrossEntropy استفاده میکنیم. و نوع بهینه ساز را گرادیان گاهشی تصادفی میگزاریم. این دو انتخاب در جواب نهایی مسئله اهمیت بسیار بالایی دارند.

```
# training
for epoch in range(num_epochs):
  ############
  # Your code
  # Start
  running_loss = 0
  correct, total = 0, 0
  for images, labels in train_loader:
      # Flatten
      images = images.view(images.shape[0], -1)

      # training pass
      optimizer.zero_grad()
      output = model(images)

      loss = criterion(output, labels)
      loss.backward()

      optimizer.step()

      running_loss += loss.item()

      # calculate accuracy
      pred = output.max(1, keepdim=True)[1] # get the index of the max logit
      correct += pred.eq(labels.view_as(pred)).sum().item()
      total += int(labels.shape[0])

  else:
    val_acc_tmp = get_accuracy(model,mnist_val)
    val_acc.append(val_acc_tmp)
    print("Epoch {} - Training loss: {} - training accuracy: {} - validation accuracy: {}".format(epoch, running_l
    losses.append(running_loss/len(train_loader))
    train_acc.append(correct / total)

  # End
```

کار اصلی training در قطعه کد بالا انجام میشود. به تعداد epoch های مشخص شده تصاویر و لیبل ها را از data_loader میگیریم. وظیفه data_loader تقسیم دیتاست است به طوری که در هر قسمت به اندازه batch_size مشخص شده از دیتاست وجود داشته باشد. (در واقع کار تولید batch ها انجام میشود.).

تصاویر به وکتورهای خطی تبدیل میشوند تا بتوان ارایه 784 تایی را به عنوان ورودی به شبکه پاس داد و خروجی شبکه که در اینجا با نام output مشخص است را بدست اورد. سایز output شبکه به اندازه Batch است(در این مسئله اندازه batch 64 در نظر گرفته شده است.). سپس output با label واقعی دیتا مقایسه میشود و loss برای آن batch محاسبه میشود.

Loss بدست آمده را به شبکه منتقل میکنیم تا وزن ها اپدیت شود. پس از اندازه گیری loss نوبت به محاسبه دقت میرسد که میتوان با مقایسه output و loss آن را حساب کرد به این صورت که هر جا خروجی پیشبینی شده با label اصلی یکسان بود یک واحد به correct اضافه میکنیم و سپس دقت برابر میشود با تعداد درست به تعداد کل دیتا.

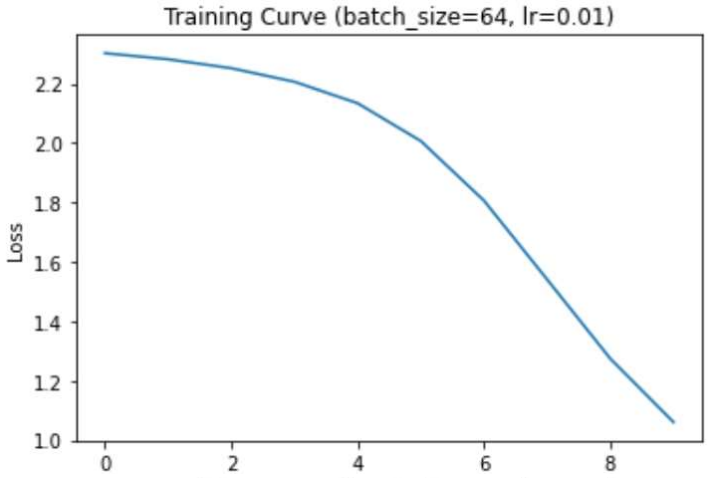
در قسمت else توضیحاتی از عملکرد مدل شامل مقدار loss و دقت برای کاربر چاپ میشود و در نهایت نمودار عملکرد مدل رسم میشود.

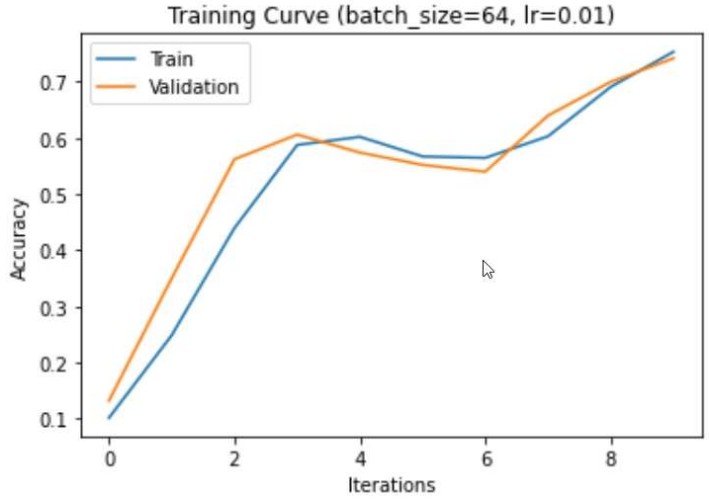
```
# plotting
plt.title("Training Curve (batch_size={}, lr={})".format(batch_size, learning_rate))
plt.plot(losses, label="Train")
plt.xlabel("Iterations")
plt.ylabel("Loss")
plt.show()

plt.title("Training Curve (batch_size={}, lr={})".format(batch_size, learning_rate))
plt.plot(train_acc, label="Train")
plt.plot(val_acc, label="Validation")
plt.xlabel("Iterations")
plt.ylabel("Accuracy")
plt.legend(loc='best')
plt.show()
```

و در نهایت خروجی مدل را در شکل های زیر مشاهده میکنید:

```
[13]: #model = ... #creat the model
run_gradient_descent(model, batch_size=64, learning_rate=0.01, num_epochs=10)
```

```
Epoch 0 - Training loss: 2.3027472496032715 - training accuracy: 0.1015 - validation accuracy: 0.132
Epoch 1 - Training loss: 2.2823048681020737 - training accuracy: 0.2485 - validation accuracy: 0.35
Epoch 2 - Training loss: 2.252242587506771 - training accuracy: 0.4395 - validation accuracy: 0.562
Epoch 3 - Training loss: 2.2065130695700645 - training accuracy: 0.5875 - validation accuracy: 0.606
Epoch 4 - Training loss: 2.1342387720942497 - training accuracy: 0.602 - validation accuracy: 0.574
Epoch 5 - Training loss: 2.0072250589728355 - training accuracy: 0.567 - validation accuracy: 0.552
Epoch 6 - Training loss: 1.8066281005740166 - training accuracy: 0.5645 - validation accuracy: 0.54
Epoch 7 - Training loss: 1.5411705672740936 - training accuracy: 0.603 - validation accuracy: 0.64
Epoch 8 - Training loss: 1.2747339755296707 - training accuracy: 0.691 - validation accuracy: 0.7
Epoch 9 - Training loss: 1.0611965097486973 - training accuracy: 0.7535 - validation accuracy: 0.742
```





مراجع

1) Nadeem, U., Shah, S. A. A., Bennamoun, M., Togneri, R., & Sohel, F. (2021). Real time surveillance for low resolution and limited data scenarios: An image set classification approach. Information Sciences, 580, 578-597.

2) https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627